articles

A Summary of the Presentations at APL Users Conference Workshop 3

Berkeley, California
April 20–21, 1971
John R. Clark
Thomas R. Dickson
William H. Marshall
Angeloa Segalla
Coast Community College District
Costa Mesa, California

Those summaries marked by an asterisk were compiled from notes and are not based on papers submitted in writing or the author's summary. Any errors or misstatements are the responsibility of the note takers.

APL as a Conversational Language – What is Missing?*

Dr. Alan Perlis
Carnegie–Mellon

Only good inventions merit improvement; APL is good and merits a lot of improvement. The elegance of APL is a bar to change and any changes should not greatly change the style and techniques of the language. APL is a natural extension of assembly language programming; it appeals to various types of thinking and is best for educational purposes. APL is conversational as a result of the nature of APL as a system. In a conversational language the programmer is in constant contact with the program. A symmetry exists between the machine and the programmer such that program data is independent of the source of the data.

What is missing in APL? The system should allow the dynamic improvement of the program. When some undefined variable is encountered, the system should ask for the value. True top-down programming should be implemented. It would be desirable to extend editing capabilities to piece a function together without terminal input. A procedure to generate programs from pieces is needed, as well as the execution of system commands within programs. There is a need for APL subsystems since one man's constant is another man's variable. An APL compiler, optimally a conversational compiler, is needed. The system should aid the programmer in flushing out incorrect programs. The programmer should be able to compute in the middle of a program. The block structure of Algol could be incorporated into APL. The powerful procedure approach should be more fully developed in APL.

A formal definition of APL is needed. APL changes should be done with care, but they must be made. They will be difficult if the system of APL is to be maintained. Today, the S/360 version of APL serves as a working standard but a fixed APL standard should be avoided at this time.

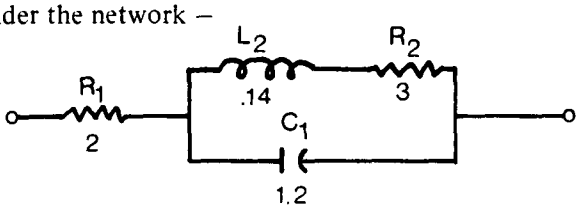
A Set of APL Programs for Use in Network Theory

Dr. Paul Penfield, Jr.
Massachusetts Institute of Technology

A set of general-purpose programs for analyzing linear electrical networks, named **Martha**, has been available to APL users at M.I.T. since September, 1970.

The programs analyze most linear "transmission-type" networks with an input and an output, as a function of frequency.

"Wiring operators" are used to form a network for simpler sub-networks. For example, consider the network –

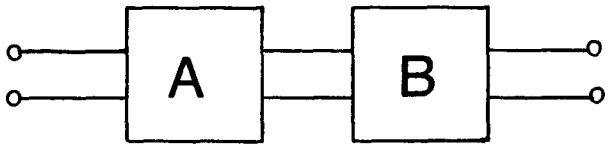


Martha describes this network as:

```
(R 2) S (((L 1.4E¯10) S (R 3)) P (C 1.2E¯6)
```

where R_1, L_1, R_1, and C_1, are elements of S and P are the wiring operators.

Martha uses the fourteen operators: P, S, WP, WS, WC, WTO, WTS, WT, WPP, WPS, WN, WSP, WSS, WROT. The figure below illustrates the operator WC.



Although Martha has not been used significantly by lower division students, it has proved fruitful for juniors, seniors and graduate students at M.I.T. The later group, for example, has used Martha to model microwave circuits and intermodulation distortion products in a slightly nonlinear circuit.

An On-Line Proof Checker Operating under APL/360, with Applications for Computer-Aided Instruction in Logic, Mathematics, and Computer Science

Paul D. Page
University of California at Los Angeles

A formal proof checker and basic theorem prover for predicate logic with equality has been implemented under the APL/360 system at UCLA. This proof-checking program operates in an on-line interactive mode and has the following characteristics:

1. The program implements the Kalish–Montague system of formal logic.
2. The input language is natural and convenient.
3. Allowed assumptions are automatically supplied.
4. Proof and subproof completion are automatically detected.
5. All necessary lines of logical inference in the Kalish–Montague system are available.
6. Logical inferences may be performed by supplying only the name of the inference rule involved; this provides an abbreviated input form.
7. Useful theorems and axioms may be recalled for use.
8. Intuitive jumps in reasoning can be made, causing the proof checker to invoke its theorem proving functions in order to resolve the logical discontinuity.

This proof checker, due to its natural input-output language, is well suited for use as a study tool by students of logic, mathematics, and computer science. It is planned that the program will be used as a basis upon which to build systems for:

1. Computer-aided instruction in symbolic logic.
2. Proving theorems of mathematics (algebra and number theory).
3. Proving correctness of simple computer programs.

The CDC Star – 100 An APL Oriented Machine

This talk covered briefly the STAR computer hardware including Physical Memory Structure, Virtual Memory Structure and Instruction set. The Sub-Options available for a typical vector instruction were explained. The APL like characteristics of the machine were indicated. Its limitations for TENSORS of rank greater than 1 were pointed out.

A comparision of the instruction set of STAR and the vector extensions of the primitive scalar functions (as defined by IBM) showed that about 60% of the vector primitive functions were directly available in the instruction set. A similar comparison with the mixed primitive functions showed about 50% of the vector forms available in the instruction set.

APL functions defined in Iversons book but not in current implementations, such as mask, mesh, maximization and minimization, which exsist as single instructions on STAR were cited. The conclusion was that more than 50% of APL functions are built into the STAR instruction set and that APL should perform well on the STAR.

Conditional Branch, APL Compiler

Dr. John Williams
Cornell

Two extensions have been added to the LRL implemetation of APL on the CDC 7600 The first extension, the monadic ϵ or execute operator as has been discussed in previous APL meetings. This extension allows:

```
A∈(2 3⍴'B+CD+E')[I;]
```

which corresponds to:

```
A←B+C  IF  I=1
A←D+E  IF  I=2
```

The second is a conditional IF operator. The IF operator is monadic, but functions as a dyadic operator.

$$E_1 \ IF \ E_2$$

will return the value of expression E_1 if the value of E_2 is not false (i.e. zero or null).

```
A←-A  IF  A<0
```

This operator allows the creation of one line recursive functions.

```
    ∇  Z←FACT N
[1]    Z←Z×FACT ¯1+Z  IF  1≠Z←N  ∇
```

Several problems are involved in attempting to compile APL programs in machine code. The fact that "A" may be a function in "F" and a variable in "G" is such a problem.

```
  ∇Z←A X
[1] Z←X ∇

  ∇ F
[1] A +B  ∇

  ∇ G;A
[1] A←1
[2] A+B ∇
```

It is possible that some restructions to the language must be imposed to facilitate compilation, however, such restrictions should maintain an upwards compatability with the full interperative APL.

A Language Machine

Zaks and Steingart
Center for Research and Management Sciences
University of California at Berkeley

A devoted time-sharing system is being developed at the University of California at Berkeley for control of behavioral science experiments. The APL language has been found to be very effective in complex decision-making simulations. Thus, the source language of the system is a special dialect of APL.

Meta–APL is a two processor, time-sharing system being developed by the Center for Research and Management Sciences, University of California at Berkeley. The first machine is a powerful microprogrammed processor with a very fast read-only memory. This processor contains an APL interpreter and a time-sharing monitor. It serves as a dedicated hardware APL processor, communicating through core with the second processor. This second processor handles all peripheral devices and provides processing capabilities such as editing, formatting, and conversion from external APL to internal APL. The system is in development and will be complete sometime in 1971.

Bulk I/0 and Communications with LTSS

Jerry L. Owens
Livermore Time Sharing System

An extension to APL is being put into an APL interpreter running on the CDC 6600 and 7600 at Lawrence Radiation Laboratory in Livermore. This extension allows a programmer to do I/0, file manipulation and system calls within his APL program. The extension consists of a dyadic I-Beam (Note: In LRL's APL character set, for a mod 33 TTY,"S is used for the I-Beam). Figure I describes the syntax and meanings for this extension.

It is intended that the create call have the effect of creating something that does not exist and return a 0 if the thing created already exists. The open call attaches the APL program to something that already exists and returns a 0 if the thing opened does not exist. Some possible contents for the character string X are illustrated in Figure II.

It will not be necessary to open or create a file or tape before reading or writing it. But the first time a file is read or written, it will be necessary to supply enough information so that the read can be performed. The information needed for a read is: the medium and name; the address (note: address for tapes means record number and for disks it means that word number); the size of the read; the mode (BCD or binary), if coming from tape; and, the data type.

L R L DYADICS "S

Result	Syntax	Type of Op.
Values Read in	1" SX	Read
Values written out	(1" SX)	Write
0 if error 1 if OK	4" SX	Create
0 if error 1 if OK	5" SX	Open
0 if error 1 if OK	6" SX	Close
0 if error 1 if OK	7" SX	Destroy
0 if error 1 if OK	-9" SX	System Call

X is a character string

Note: Execution stops if a read or write fails for any reason

Figure 1

EXAMPLES OF CHARACTER STRINGS FOR "S

'File ( ) Add ( ) Size ( ) Type ( )'
'Tape ( ) Add ( ) Mode ( ) Size ( ) Type ( )'
'CTLR ( )'

'CTLE ( )'
'ELF ( )'
'TTY'
'System ( ) In( ) Out ( )'
'File ( ) Size ( )'
Any executable statement may appear inside the ( )
The possible values of the argument to type are:
0 = Log
1 = Char
2 = Integer
3 = Float

Figure II

Generalized Lists and Other Extensions*

Jim Ryan
Burroughs Corporation

With the introduction of lists as a data type in the APL language, it would be possible for the APL user to:

a. Save and modify subscripts while executing a program.
b. Save and modify arguments to functions while executing a program.
c. Work with structured data as found in COBOL and PL/I.

The lists may be described using the notation for trees and would be represented in a manner simular to the current use of subscripts. Operations on lists would require the introduction of new operators.

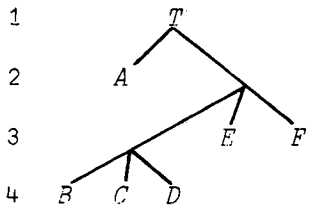


Figure 1

Figure 1 represents a 4 level tree which would be defined by the APL user as:

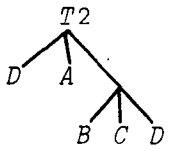
```
T←(A;((B;C;D);E;F))
```

It is possible for any fork to have many branches and any leaf can be obtained by using an access vector which indicates which branches to take.

```
B  ←→  T[2 1 1]

E  ←→  T[2 2]
```

Additional trees can be constructed as in Figure 2.




```
T2←T[2 1 3; 1; 2 1]
```

Figure 2

The operator ⍕̈ would yield the shape of the tree.

```
2  ↔  ⍴̈[1]T          ↔
```
Indicating level 1 of T has two branches.

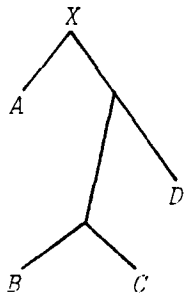
```
(⍬;3)  ↔  ⍴̈[2]T      ↔
```
Indicating at level 2 the left branch of T has terminated and the right branch is the start of a three branch subtree.

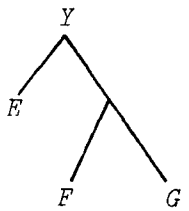
```
(⍬;(3;⍬;⍬))  ↔  ⍴̈[3]T
(⍬;((⍬;⍬;⍬);⍬;⍬))  ↔  ⍴̈[4]T
```

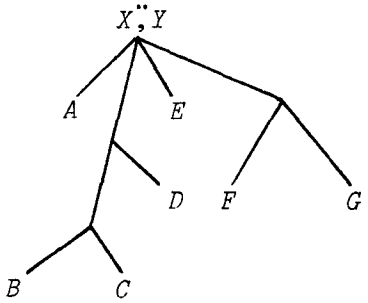
By use of the ¨ additional list operators could be defined.

```
2 1 3  ↔  T⍳̈D  ↔
```
The first occurance of *D* in *T*.

X + Y would add corresponding elements of X and Y with leaf F replicated to correspond to leafs B and C. An error simular to a length error would occur if simple replication is not possible.







Two new symbols are proposed to cover the problem of the dequote operator.

Monadic ⊥ for Dequote
Monadic ⊤ for Enquote

```
11  ↔  ⊥'3+8'

'11'  ↔  ⊤3 + 8
```

In order to avoid repeating subscripts when incrementing an element of an array, the notation

```
X[--;--;--]+←5
```

replaces

```
X[--;--;--]← X[--;--;--] + 5
```

LRL APL Implementation on CDC 6000–7600*

Dr. Ned Dairike
Livermore Radiation Laboratory

The preparation for the STAR computer, included the implementation of APL on the existing CDC6000-7600. The implementation runs under the Livermore Time Sharing System. Each user has his own copy of APL and is provided with a variable workspace. The size of the workspace may be determined at initial sign on, by system command and by dynamic allocation. The maximum size of the workspace is the size of the computer.

The existing 200 teletype terminals forced the adoption of a new character set. Some substitutions are: "F for Minimum, "C for Maximum, "I for iota, "$+ for Nor, "U for grade up, "D for grade down. This character set has been found to be usable.

Some time statistics were obtained with the same function running on the CDC vs an IBM model 50 and model 91 computer. Various tests on the Ackerman function gave a ratio 1:13.1:1.43 for the CDC, model 50, and model 91, respectively. Time statistics dealing with matrix inversion yielded a ratio of 1:17.2:1.67.

Time Sharing APL for IBM 1130 Systems

Thomas P. Daniell
IBM

APL/1130–MTCA is a program for IBM 1130 systems which provides APL service for multiple terminals. Access to the program is provided by IBM 2741 Communications terminals. The program requires at least 16K words of core storage and at least one 2310 disk storage drive.

Most of the APL/1130–MTCA program is disk resident. Portions of the program are transferred to core storage as required by user activities. All of the primitive functions supported by APL/360–OS are supported by APL/1130–MTCA. Arrays up to rank eight and having up to 32767 elements are supported.

All user workspaces (including active workspaces) are disk resident. Portions of a user's active workspace are brought into core storage as required by his activities. Each terminal user is allocated a fixed area on one of the library disk packs. This allocation is shared by his active workspace and by the workspace in his library. The size of his active workspace may be changed by the user or may be made to vary dynamically.

APL on the Honeywell 635

Norman Glick and Richard Schrader
National Security Agency

APL has been implemented for the Honeywell 635 computer. Most of the primitives have been implemented and some special system commands provide linkage to other subsystems within the GECOS operating system. The computer system is a Honeywell 635 with 256 K of core, two DSU 270 disk units and one MDS drum unit. Thirty IBM 2741 typewriter terminals are hard-wired to the system.

The implementation language is Fortran IV, with some assembly language routines. The ASCII character set is used rather than the APL character set since the terminals are used most often for other TSS subsystems. The APL operators are selected from the symbols, upper case letters and a few lower case reserved words. The choice of characters was based on four factors. First, some of the APL characters are in the ASCII set. Second, the same keyboard positions are used in some cases. Third, some ASCII characters and APL characters are similar in appearance. Fourth, a mnemonic letter is used if needed. When necessary, a two or three letter keyword is used. A set of system commands, similar to APL/360 has been implemented. Three of these are unique to APL/635. The)INCLUDE command provides for the input to APL from ASCII files. The)DDT command provides entry to a powerful debug package. The)CMDINT provides a link to the rest of GEOs TSS from APL.

The size of APL/635 is 36 K and a 3000 word table represents the size of the workspace.

A Micro–Programmed Implementation of an APL Machine

A. Hassitt, J. W. Lageshulte, L. G. Lyon
IBM Corporation, Palo Alto Scientific Center

The standard IBM 360 is a micro-programmed machine. This means that the hardware can execute certain single operations and a micro-program (the 360 emulator) uses a sequence of these simple operations to execute a single 360 instruction. We have written an APL emulator which runs on the Model 25 hardware and which directly executes APL code. There is a simple translation from external APL code to internal APL machine form; this process is not compilation, it is simular to an assembler process. The microcode handles the complete execution of the following: statement, scan and syntax analysis; space management and garbage collect; function call and return; most of the scalar operations on scalars, vectors and arrays, assignment, go to, monadic rho and iota, diadic rho, scalar and vector subscripts and scalar or vector

compress and expand. The machine does support the full language; if the microcode encounters an operator which is not implemented in microcode (for example the dyadic ρ) then it automatically calls an APL system function which does the appropriate function and returns to the microcode. We have written a simple one user supervisor/translator which is written in APL and which ran on the APL machine. The machine has been fully operational since June, 1970.

Comparative times in micro second for the operator $A \leftarrow B + C$ is shown in the table below.

No Elements	APL–25	ALC–25
Scalar	682	298
Vector 5	2385	1491
10	3985	3926
20	7185	7796
40	13585	15536
N	785+320N	56+387N

APL/700 – An APL Implementation for theBurroughs 6700 and 7700

Jim Ryan
Burroughs

The APL/700 implementation is based on interactivity between the user and the workspace. Optimum error recovery is provided which allows the user to restore the workspace environment to its initial state prior to the execution of a function which failed to complete execution. Phantom length errors are weeded out when the result of the next operation does not require more elements than the shortest vector.

Environmental protection and error recovery is by the use of "soft" and "hard" functions. A soft function is a function which has just under-gone some form of editing. Several executions without modification of a soft function, will cause it to evolve into a hard function. While in a soft state, assignments to global variables are kept in temporary locations, until the function has completed execution.

A length error which would be generated in the major implementation of APL by the statement $X \leftarrow 3 \times 2 \uparrow A + B$

$$(\rho A) \neq \rho B$$
$$2 \leq (\rho A) \lceil \rho B$$

would not appear, due to the list structure recognizing that only two elements of the addition is required for the multiplication.

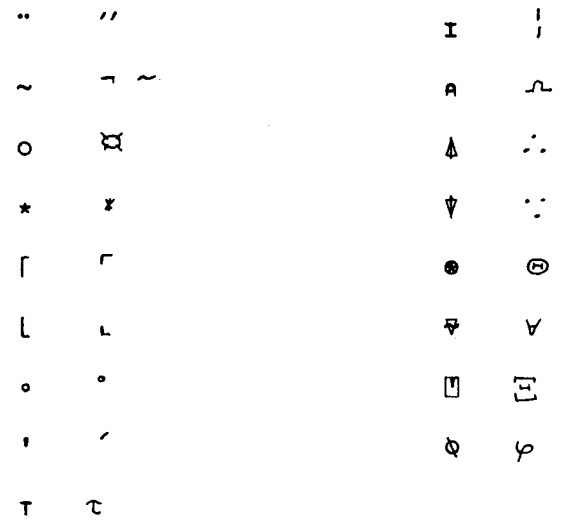
Statements entered from the terminal are checked for visual accuracy before being placed in the code stream. If an error is found, the user has the ability to edit the statement even though he may be in calculator mode at the time. The user is given the option of editing the statement, providing a value or aborting and throwing away side effects generated when executing a statement $X \leftarrow A + B$ where B is undefined.

An 8–Bit ASCII Code

Dr. John Fletcher

A subcommittee of the X3L2 group is currently studying an 8-bit code which may become the new ASCII standard.

Many of the current APL symbols are included. For those APL characters not included in the new ASCII proposal, the following substitituions are proposed.



One apparent advantage to this new convention, is the elimination of the current overtrike characters. However, it should be noted, a large computer manufacture did not adopt the current 7–bit ASCII standard and may not be completely overwhelmed by this new 8-bit proposal.

The MRX 1240 Communication Terminal and 1270 Transmission Control Unit*

David Sant
Memorex Corporation

The MRX 1240 was designed from the ground up to be a computer communications terminal. It is a line printer device capable of speed up to 60 characters per second. The full APL S/360 character set is supported as well as the extended APL characters of Burroughs /700 implementation. A pin feed option is available and forms of eight part paper may be handled by the terminal. A 1270 transmission control unit which is plug compatable with the S /360, is required to support the terminal. Additional announcements are expected during the month of May.

A Plotter of APL

Mike Dayton
Time Share Peripherals

A computer-controlled plotter with a rather high degree of accuracy was demonstrated by Mike Dayton of Time Share Peripherals.

The implications for such device are far reaching for APL users.